

A large collection of various objects, including plastic bottles, metal pieces, wood, and other debris, arranged on a light surface. The objects are scattered across the entire frame, creating a dense and diverse visual field. Some objects are recognizable as everyday items like a green plastic jug, a blue plastic bottle, and a metal mesh tray, while others are more abstract or natural, like a piece of wood, a piece of bone, and a piece of fur.

# Introductie Object-Oriented Programming

1

[edwin.vanouwerkerkmoria@hku.nl](mailto:edwin.vanouwerkerkmoria@hku.nl)

# Over deze lessen

Flink veel programmeer werk

*Programmeren (redelijk) nieuw? Lynda!*

“Learning C++” van Peggy Fisher

Module is 2 EC = 56 uur

8 (werk)colleges

*ong. 4 uur/week eigen werktijd*

# Opdrachten per week

## Geen aparte eindopdracht

Je beoordeling gaat op basis van de opdrachten die je per sessie krijgt

*Opdrachten de ochtend vóór volgende keer bij mij inleveren voor beoordeling!*

*Dus: sessie op donderdagmiddag? Inleveren voor woensdag 12:00*

We bespreken de oplossingen/problemen gezamenlijk.

# Over deze lessen: inhoud

## Basiskennis C++

*Classes, methods,*

*stack vs. heap, pointers, references, pass-by-value*

*'The Big Three': constructor, copy-constructor, assignment operator*

*Basisprincipes OO: inheritance, polymorphism, overloading, abstract classes*

*IO streams, stream operators*

# Object-Oriented programming?

Anders dan 'gewoon' programmeren: Manier om een (geprogrammeerd) systeem op te bouwen uit objecten.

In plaats van losse 'acties' (subroutines, functies), gecombineerd in Objecten

In 1 pakketje: **gedrag**, en '**state**' (data)

gedrag: 'methodes' (*geen 'functies'*)

state: 'fields' of 'attributes'

Objecten kunnen **hergebruikt** worden

Definitie van een Object: **Class**

# Object-oriented programming

Het is nog steeds 'gewoon' programmeren:

*variabelen*

```
int totaalAantalPunten = 15;
```

*methodes / functies*

```
float getInventoryWeight() {  
    // rest van de code  
}
```

```
void addToInventory(InventoryItem thing) {  
    // rest van de code  
}
```

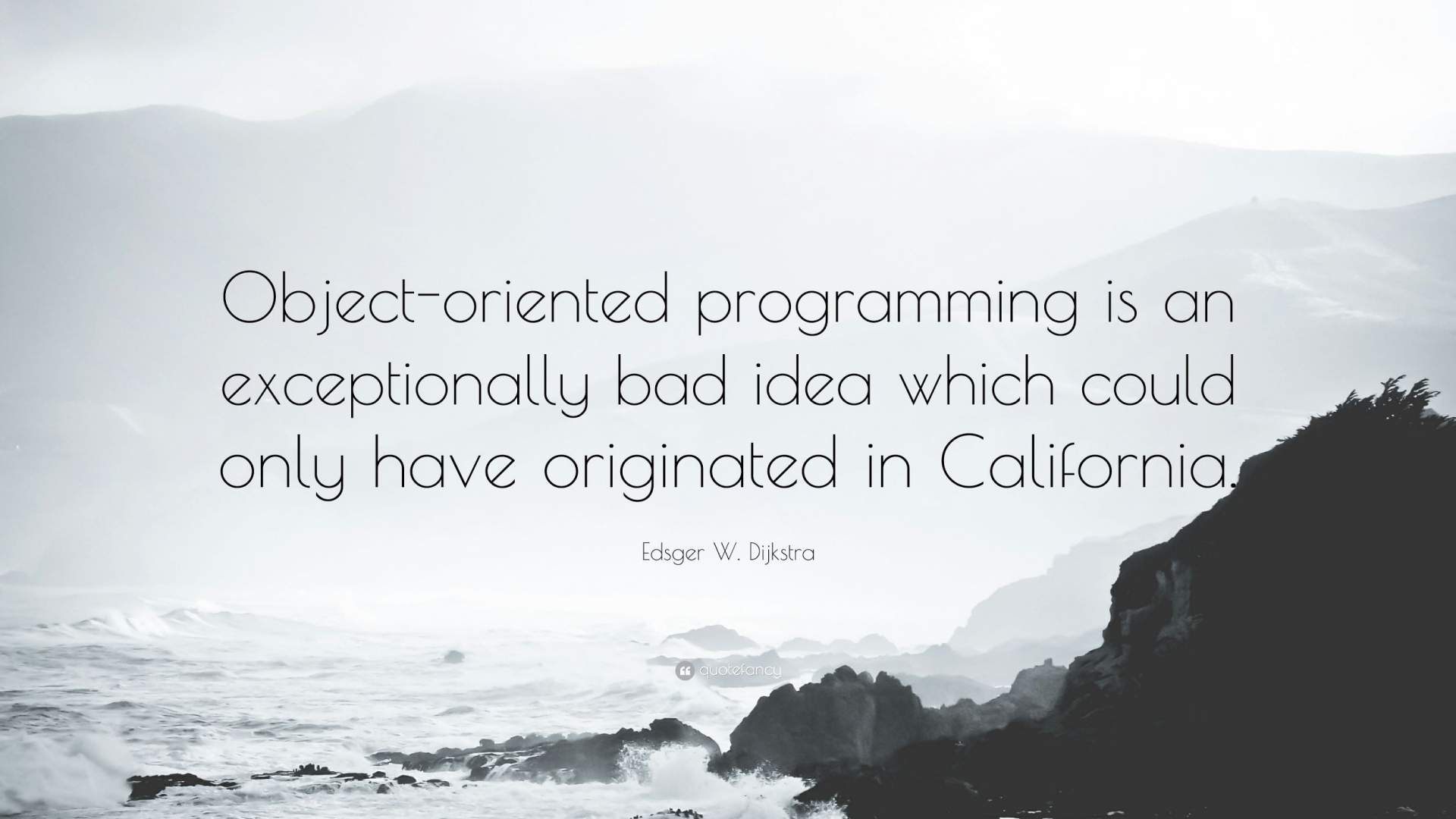
# Object-georiënteerde talen

Modern

*Java, C#, Python, PHP, Ruby, Scala, Objective-C, Swift,*

Ouder

*C++, Smalltalk, Eiffel*



Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger W. Dijkstra

“ quote fancy



# C++ : “C met objecten”

Gebaseerd op oudere programmeertaal C (eind jaren '70)

Veel embedded software in C, want compact, snel, dicht tegen hardware

*Arduino: aangepaste versie van C & C++*

Begin jaren 80: Objecten en Classes toegevoegd aan C

C++ is in concept 'laagje over C'

Je kunt nog steeds prima C & C++ mixen

# C++: definitie van een **class**

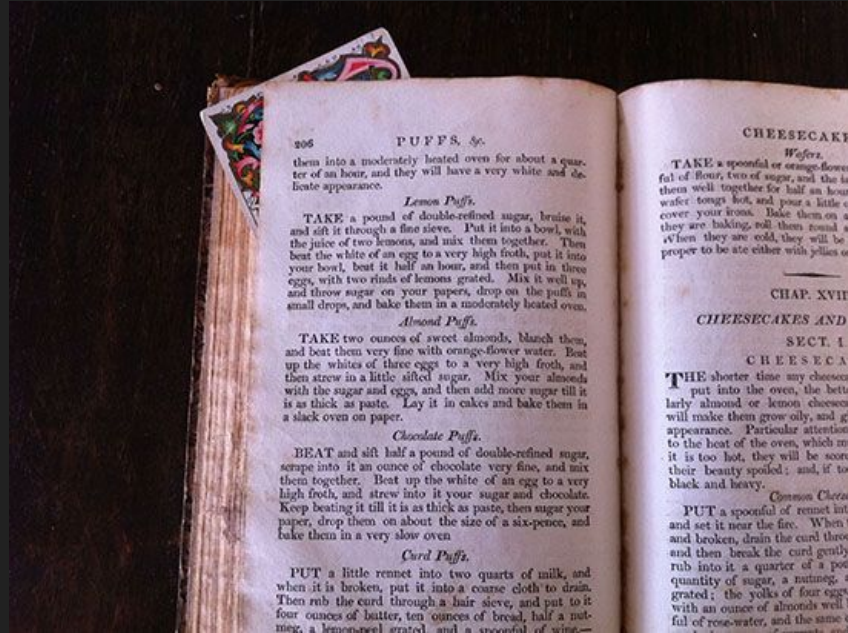
HEADER file (.h)	IMPLEMENTATIE file (.cpp)
<pre>class Rectangle {     private:         int width, height;         float transparency;      public:         Rectangle(int w,int h);         void setTransparency(float a);         int area();         void draw(); }</pre>	<pre>Rectangle::Rectangle(int w,int h) {     width = w;     height = h; }  void Rectangle::setTransparency (float a){     transparency = a; }  int Rectangle::area() {     return width*height; }  void Rectangle::draw() {     ... }</pre>

# Maak een **object** op basis van een **class**

```
void startMijnBakkerij() {  
    Taart mijnTaart = Taart();  
}
```

*Een Class **beschrijft** wat een **Object** allemaal kan. Een Class zelf is nog niks, en kan nog niks...*

# Object-oriented programming: Class vs Object



Oftewel: de *class* is het concept, het *object* de realisatie...

# Een class = een type!

```
float eindCijfer = 5;
```

```
int eindCijfer = 5.7;
```

```
string studentNaam = "Pietje Puk";
```

```
int studentNaam = "Pietje Puk";
```

```
Taart mijnTaart = Taart();
```

```
Bankrekening rekening = Taart();
```

# Een class = een type!

✓ `float eindCijfer = 5;`

✓ `int eindCijfer = 5.7;` ⚠ `// wordt afgekapt naar 5`

✓ `string studentNaam = "Pietje Puk";`

✗ `int studentNaam = "Pietje Puk";`

✓ `Taart mijnTaart = Taart();`

✗ `Bankrekening rekening = Taart();`

# Het startpunt van je programma : `main()`

Ieder programma heeft een beginpunt nodig. Waar moet de computer beginnen met het runnen van je code?

In C/C++ heet dit beginpunt de `main()` functie (*functie! geen methode!*)

De `main()` functie ziet er meestal zo uit:

```
int main(int argc, char* argv[]) {  
    // start van je programma  
}
```

De parameters zijn optioneel, dus `int main()` mag ook!

# OO-analyse

Bedenk welke objecten (concepten) er in je programma voor gaan komen  
*goed begin: alle zelfstandige naamwoorden*

- identificeer objecten
- bepaal gedrag per object
- bepaal welke data (state) ieder object nodig heeft
- bepaal relaties tussen objecten

Voor 't weergeven van deze informatie is een teken-taal: **UML**





# Codingstyle

Code schrijf je in eerste instantie voor jezelf, in tweede voor je collega(s), en pas op de laatste plaats voor je compiler.

- Schrijf *leesbaar*: duidelijke variabele- en methode namen
- Formatteer consistent (je IDE helpt)
- Voeg commentaar toe

Haakjes boeien me niet. Wel een paar basic C++ conventies (*geen C# stijl dus!*)

- Classes beginnen met een **HOOFDLETTER**
- methods beginnen met een **kleine letter**
- variabelen beginnen met een **kleine letter**

# Opdracht voor volgende keer

Modelleer het lesrooster

- welke objecten spelen volgens jou een rol?
- wat zijn de attributen (variabelen) van die objecten?
- wat zijn de relaties tussen die objecten?
- wat voor gedrag moeten die objecten hebben?

Maak C++ classes voor de objecten die je bedacht hebt.

*Je hoeft de methodes niet te implementeren - geef ze alleen een naam, en commentaar wat ze zouden moeten doen.*

*Declareer wel de attributen (variabelen) die je nodig denkt te hebben.*